68ØØFORTH

Reference Manual

PROGRAMMA

Software
Program
Products

6800.001

1. INTRODUCTION.

6800FORTH is a unique threaded language that is ideally suited for microprocessor systems. Programs written in 6800FORTH are compact; i.e. in 5K to 6K bytes, the user may have the interactive 6800FORTH compiler/interpreter running stand-alone using the system's monitor for I/O, and other run-time routines , plus an assembler in 6800FORTH, cassette memory software, and a text editor. Not only does all of this fit into the 5K to 6K bytes (4K of which are written in 6800FORTH), but also, it runs in the same space with no additional symbol table area, overlays, swapping, or use of any other software.

while 6800FORTH gives all of the conveniences of interactive interpreters, it is also very fast. For most applications, the run-time overhead is 70 to 100 percent for microcomputers, as compared to 1000 percent or more which is common for interpreters such as BASIC. Number crunching applications in 6800FORTH may take much longer, however, if 6800FORTH is not fast enough, the user may choose to use the system's own assembler to re-code inner loops.

One of the best advantages of 6800FORTH over other programming languages is that software development times are cut in half or much more over assembly language programming. The programming in 6800FORTH is entirely done in a structured manner (there is no GOTO), and the resulting code is re-entrant and can be designed for PROM.

The 6800FORTH implementation of the FORTH language requires a machine configuration that contains both an input keyboard device and an output display device. A recommended memory configuration is 16K bytes, however, the system will work satisfactory in 8K bytes. A disk device is nice for storing 6800FORTH source programs, but the audio cassette recorder (or similar sequential unit) suffices to simulate the virtual memory storage of source programs. 6800FORTH works well with a CRT video display unit, so hard copy is not necessary.

The FORTH language has existed for several years, and is used commercially in a number of installations. Until recently it has been priced far out of the reach of the amateur hobbyist. Most computer professionals have never heard of it. The FORTH language, however, is in the public domain, and 6800FORTH is the first implementation on the 6800 Microprocessor unit.

The original FORTH language was first developed by Charles H. Moore at the National Radio Astronomy Observatory under contract with the National Science Foundation. A paper in the Journal of Astronomy and Astrophysics Supplement (1974, 15, 497-511) titled "FORTH: a New Way to Program a Mini-Computer", by Charles H. Moore, describes the original specifications and description of the FORTH language.

2. OVERVIEW.

The basic element of the 6800FORTH system is termed a "word", which is roughly comparable to a subroutine. A 6800FORTH word, when referenced (or executed), causes an action or sequence of actions to be performed. Therefore, when a word is executed, a subroutine is called and the various actions indicated by the subroutine occur. Before a word can be executed, it must have been previously defined and stored in the 6800FORTH "dictionary". The dictionary is a linked list of words together with their meanings or actions. The actions may be expressed as machine-language instructions or as a sequence of other words. The 6800FORTH dictionary initially contains around 200 words, which are referred to as the "standard vocabulary". Some of these words can be used to define new words. Writing a 6800FORTH source program consists of defining a series of new words in terms of the old definitions.

A 6800FORTH user at the keyboard terminal may type words into the computer. Any sequence of characters may be used to define a word. The only reserved characters are those that have special meaning for the machine environment that's being used. Otherwise, any combination of letters, numbers, and special characters can be used in defining the name of a 6800FORTH word. A word must be separated from other words by a deliminator character. The deliminator character is normally a space or blank. Input from the keyboard terminal is "buffered" by the 6800FORTH system, and control passes to the system for execution when the "Carriage Return" key is depressed. For example, the input stream:

```
7 3 + . CR
```

will cause the numbers 7 and 3 to be added together and a result of 10 (assuming base 10) to be printed on the output device. The 6800FORTH system will then do a "Carriage Return" and "Line Feed" and proceed to prompt the user for further input.

In order to conserve computer memory, not all of the characters in a name are stored. In the 6800FORTH implementation, a name is recognized on the basis of the first four (4) characters.

Reverse Polish Notation (RPN) and Last-In First-Out stacks (LIFO), such as those used in Hewlett-Packard calculators, are used in the 6800FORTH system. Therefore, to further explain the previous example in detail: The number 7 was pushed onto the stack, and it was followed by the number 3. Both numbers were then added together and "popped" off the stack by the previously defined word "+". The result of 10 is "pushed" onto the stack by the "+" operation also. The word "." then "pops" the stack to its initial condition and prints the number 10 on the output list device.

If a word that is typed in the input stream cannot be located in the 6800FORTH dictionary, the system attempts to treat it as a number. If this is possible, that is, if the word is actually a number in the proper format and base, then the number is converted to binary and made available for further processing. If the word cannot be interpreted as a number, or if conversion is not possible, then 6800FORTH will issue its standard error message: ? (a question mark).

Words can be added to the 6800FORTH dictionary in several ways. As with any programming language or notation, the fastest route to fluency is through examples and hands-on usage. Therefore, if the input stream consists of:

```
2 VARI VALU
```

6800FORTH defines a new word in the dictionary called "VALU", which is a 16-bit (two byte) variable, whose value is preset to "2". Remember that 6800FORTH only looks at (and only remembers) the first 4 characters of a word. The input stream:

```
2 VARIABLE VALUE
```

would produce exactly the same results. Continuing with the same example, the input stream:

```
VALU @ .
```

causes the address of "VALU" to be pushed onto the stack. The "@" then replaces the address on the stack with the contents found at that address. The "." causes the entry on the stack to be printed on the output device. Hence, a two (2) would be printed on the output device.

Words that are already in the 6800FORTH dictionary may be used to form other new words using the ":" word. For example:

```
: ? @ . ;
```

This input stream defines a new word called "?" which when executed causes the word "@" and the word "." to be executed. The ";" word is the 6800FORTH word that terminates the definition mode. With the above new word, the user can now input:

```
VALU ?
```

which will cause the value of the variable "VALU" to be printed on the output device. In this case a two (2) would be printed.

The words that compose the "STANDARD VOCABULARY" are listed in the section titled "standard VOCABULARY". The actions of the Standard Vocabulary words are also explained in that section.

Input to the 6800FORTH system can also come from a block buffer instead of the keyboard. This buffer normally contains ASCII characters that have been previously stored on a mass storage device.

3. STACKS.

Numbers and other data are normally handled through the 6800FORTH "Normal Stack" (parameter stack). This is a "push-down" stack that uses the "Last-In First-Out" (LIFO) technique. A Push-Down stack is a storage management structure in which a new data item may be stored (pushed) on top of older data items. The item on top of the stack may be retrieved by "popping" the stack.

One advantage of a "push-down" stack is that fixed storage locations in memory need not be assigned for temporary data. Hence, storage is conserved and the programmer's "bookkeeping" effort is reduced.

Most 6800FORTH words, which operate on data, accept their data from the normal stack, operate on them, and then push the results back onto the stack. Therefore, arithmetic expressions must be specified in "Reverse Polish Notation". (i.e. with operands preceeding operators).

6800FORTH users have the option of specifying the top of RAM memory, and defining the locations of the FORTH dictionary, the "Return Stack", the "Normal Stack", and the buffers. It is recommended that the default values be used, because 6800FORTH will only check for overflows of these areas if they are in their default locations. By specifying a different location for any one of these areas, the user will cause 6800FORTH to bypass the boundry checking feature. (The top of RAM memory can be specified without cancelling this feature).

There are three (3) stacks used by the 6800FORTH system. All of them use the Last-In First-Out (LIFO) technique. Assuming the defaults are used for memory allocation, the "Normal Stack" is variable in length (depending on the RAM size available); and it grows downward toward the FORTH dictionary, which in turn grows upward. The FORTH word "SP" (stack pointer) is a constant that points to the address of the top (current) stack value. This address points to the most significant byte (MSB) of the 16 BIT stack value.

The "Return Stack" is fixed in length (2 pages) and it originates on the same location as the normal stack. It however, grows upward (it's still referred to as a "push down" stack because of its LIFO technique). The FORTH word "RS" (Return Stack) is a constant that points to the address of the top (current) Return Stack value. This address also points to the MSB of the 16 BIT value. The Return Stack is used primarily by the FORTH system for loop processing.

The third stack is the "Hardware Stack" used by the 6800 Microprocessor. It is located at location $01FF and it grows downward toward low memory. Its location cannot be changed by the 6800FORTH user. This stack is not normally used by 6800FORTH programmers, so there is no FORTH word that contains its address.

4. DICTIONARY.

The 6800FORTH dictionary is a linked list of words. The dictionary normally begins from the end of the 6800FORTH nucleus and grows toward high memory. The dictionary contains all of the 6800FORTH words available to the user.

There are three (3) groups of information that can be found within each word in the dictionary. The first four bytes of any dictionary word contain the name of the word in ASCII code. The Most Significant Bit of the first byte of the word may be set to one (1) to indicate to the 6800FORTH system that this is an immediate word. Of all the dictionary words, immediates are those that are executed directly when found in the input stream. Names are considered equivalent if their first four characters are the same. Names that have less than four characters are automatically padded with spaces or blanks.

The next two bytes of a word following the name contain the address of the first byte of the previous dictionary entry. These two bytes are used to link the dictionary. The link address of the first word in the dictionary is set to zero (0000), and indicates the beginning of the dictionary. This is also the end of the chain of linked dictionary words, because the dictionary is searched backwards (from the last word entered to the first). These first six bytes (the name and the link pointer) are commonly referred to as the "Header".

The remaining bytes of an entry in the dictionary consists of machine language code, which is really a subroutine. Hence, the word is executed (by the FORTH nucleus) by doing a Jump to Subroutine (JSR) to the first byte of machine code. The machine language code normally terminates with a Return from Subroutine (RTS) instruction.

For example:

```
: ABC CLR CR ;
```

will generate the following dictionary entry:

ADDRESS	CONTENTS	COMMENTS
1000	$41	"A" Header Start
1001	42	"B"
1002	43	"C"
1003	20	" " padding of a blank character
1004	0F	MSB of link to next word header
1005	E0	LSB of link to next word header
1006	BD	JSR instruction OP code
1007	0A	MSB of address of CLR word subroutine
1008	00	LSB of address of CLR word subroutine
1009	BD	JSR instruction OP code
100A	0B	MSB of address of CR word subroutine
100B	FF	LSB of address of CR word subroutine
100C	39	RTS instruction OP code

5. BLOCK STORAGE.

Most practical applications of the 6800FORTH language require an auxiliary storage device. "Floppy Disks" are usually preferable, however, sequential magnetic tape cassettes suffice for the average user.

Block storage is used as a "Virtual Memory" scheme, where one may store data in blocks when there is insufficient space in the computer's main memory. Blocks are suitable to store large amounts of data. Normally, in the 6800FORTH system, this data is the source text.

Blocks are usually called "Screens", and are numbered sequentially beginning at one (1). The name "Screen" comes from the use of a Memory Mapped CRT screen for the block storage buffer. 6800 machines that have this feature actually store the source data (and read it in) using the CRT screen's memory locations. This allows source editing using the curser controls. Two large buffers (Buffer 0 and Buffer 1) are set aside for those 6800 machines that don't have the feature, to allow editing.

6. KERNEL.

The Kernel (nucleus) of the 6800FORTH system is an assembly language program whose basic function is to provide the capability of starting and adding new word entries to the dictionary. Some of the tasks that are performed by the Kernel of the 6800FORTH system are:

1. Initializing the system
2. Buffering the input from the keyboard
3. Searching the dictionary
4. Converting ASCII input to numbers
5. Parsing the input buffer
6. Executing words in the dictionary
7. Adding words to the dictionary
8. Checking for errors

The Kernel is located at a fixed location in memory and is usually followed by the dictionary. The dictionary grows toward high memory. The kernel uses temporary storage in Page Zero (0) of main memory.

7. DEFINING NEW WORDS.

The distributed 6800FORTH system comes with about 200 words defined in the dictionary. These words provide the functions that are commonly needed by most application programs. Programming in the 6800FORTH language actually consists of defining new words, which draw upon the existing vocabulary, and which in turn may be used to define even more complex applications.

6800FORTH provides a number of ways to define new words into the dictionary. The language even provides a facility for defining words whose function is to define words. There are four common ways that may be used to define words using the standard system.

The word ":" (colon) is used to define other 6800FORTH words in terms of existing words in the dictionary. Colon definitions are usually machine independent, since each refers to code definitions or other colon definitions. An example of a colon definition is:

```
: NEW CLR 1234 . CR ;
```

Here the word "NEW" is defined as a sequence "CLR", 1234, ".", and "CR". The words are assumed to be present in the dictionary at the time the definition takes place. The number (1234) will cause code to be generated that will place it onto the stack when the word "NEW" is executed. Semicolon ";" is a word that indicates the end of the definition.

The words "[" (left bracket) and "]" (right bracket) allow the user to define words whose actions are expressed directly in machine (assembly) language. The words that are used within the brackets are dictionary words that cause the binary machine code to be added onto the dictionary. The names of these words have been conveniently chosen to closely resemble the machine's assembler mnemonics. For this reason, the words that are used within the brackets are machine dependent, but they give the programmer the means to achieve maximum possible speed of execution. The brackets are two of several words that can only be used within a ":" and ";" definition. For example:

```
: KILL [ 0 LDX # 0 CLR, ,X INX FB BRA ;
```

Here the word "KILL" is being defined as a sequence of machine instructions after the bracket word is detected. if "KILL" was subsequently executed, it would clear a large portion of memory.

Constants may be defined through the word "CONS". For example:

```
1234 CONS X1
```

defines the 6800FORTH word X1. Whenever X1 is executed, it will push the constant 1234 onto the stack. The use of X1 in an input stream would create fewer machine instructions than the use of the number 1234. However, both methods produce the same results.

Data may be stored in named locations as well as on the stack. The named locations are in the dictionary. This is done by using the 6800FORTH word "VARI". As an example:

```
1234 VARI X2
```

defines the word X2 which is the name of an address that contains the initial value of 1234. When X2 is executed, the address of the value 1234 will be placed onto the stack. Other 6800FORTH words are available that can either fetch the value from the address on the stack or change the value at that address.

The difference between "CONS" and "VARI" is that "CONS" defines a word which represents a value. "VARI" defines a word that represents an address.

8. INPUT / OUTPUT.

Under normal operation, 6800FORTH acquires its input for execution from the keyboard. The system is usually idle and waiting for the user to type a complete line of words. When this is done, the system interprets the line, tries to execute the valid words, and then proceeds to prompt the user for more input.

6800FORTH may also take its input from the Block buffer. The word "LOAD" causes the system to read a screen from the mass storage device and load it into the Block buffer. The input that is read is called a screen. The user should enter the desired screen ID onto the stack prior to executing the word "LOAD".

The data in the Block buffer can be edited and then executed just as if it had all been keyed in at the keyboard again. The word "EXEC" causes the input to come from the Block buffer instead of the Keyboard buffer. The input always starts from the beginning of the Block buffer and continues until the first occurance of the word ";S". The ";S" word stops the scan of the Block buffer. Any data found in the buffer after the ";S" is ignored. The ";S" also switches the system back into the keyboard mode of input.

The data in the Block buffer can be written to the Mass Storage device with the word "KEEP". The user should first ready the Mass Storage device, then put onto the stack the ID of the screen to be written. The word "KEEP" is then placed in the Block buffer immediately after the last line of good text in the buffer. The "KEEP" word clears the Block buffer to spaces beginning with it's own position in the buffer. It then places, in the buffer, at the position it occupied, the ";S" word. The buffer is then written to the Mass Storage device with the ID from the top of the stack. The ";S" is placed at the end of the data in the buffer as a convenience for the subsequent "LOAD" and "EXEC" of the screen.

9. CONDITIONAL BRANCHES.

6800FORTH provides several techniques for controlling the flow of program execution. The methods described in the following paragraphs and the examples must be used within ":" (colon) definitions. All of these definitions are immediate words. (i.e. they are executed immediately - during the compilation - and cause machine language code to be added into the dictionary. It is undesirable to add anything to the dictionary unless a word is being defined).

The simplest conditional branch is specified through the use of the words "BEGIN" and "END". The "BEGIN" - "END" construct is useful for program loops, when the loop termination condition can be expressed by leaving a zero or non-zero value on the stack. The "END" word tests the stack value and if it's zero the loop is repeated. For example:

```
: EX 5 BEGIN 1 - DUP NOT END DROP ;
```

First the value 5 is pushed onto the top of the stack. The word "BEGIN" indicates the beginning of a loop. Everything between the words "BEGIN" and "END" will be repeated until the word "END" finds a non-zero value on the stack. The value on top of the stack is always removed by the word "END". The first thing the loop does is push a 1 onto the stack. Then the 1 is subtracted from the 5 by the word "-". The only thing on the stack now is the value 4. The word "DUP" duplicates the top value on the stack. This is necessary because the word "END" is going to remove one value. So now, after the "DUP", the stack contains two values of 4. the "NOT" word switches the top value on the stack to 0 (had it already been 0, NOT would have switched it to 1). The "END" word tests the top value on the stack for 0 and then removes that value. The stack now contains only the value 4. Since the "END" word found a zero value, the loop is repeated. This process will continue to decrement the value on the stack until it reaches zero. When it becomes zero, the NOT word will switch it to 1. When "END" finds the 1, its search will be satisfied; it will remove the 1, and the word after the "END" will be executed. The word "DROP" removes the top value on the stack. It is used in this example to remove the residual zero (0) left on the stack. It's important to leave the stack as you found it. Remember, none of this will happen until the word "EX" is subsequently found in the input stream and executed.

An endless loop can be created by the following:

```
: X1 BEGIN 0 END ;
```

Other words can be placed between the "BEGIN" and the "0" to give the endless loop more purpose. Once the word EX1 is executed, it can only be stopped with the Non-maskable Interrupt key. The program must then be "Softstarted".

6800FORTH contains a looping facility that is very much like the

FORTRAN DO-LOOP construct. The words DO, LOOP, and +LOOP are used to define the FORTH DO-LOOP facility. The following example will illustrate the use of these new words:

```
                    : EX2 4 0 DO I . LOOP ;
```

When the word EX2 is executed, the constants 4 and 0 will be pushed respectively onto the stack. The word DO uses these top two values as the limit and the initial index of the loop. These numbers will be removed form the Normal stack and pushed onto the Return stack. Then the words after the word DO are executed. The word I copies the index value from the Return stack and pushes it back onto the Normal stack. (The index value is now on top of both stacks). The word "." (period) removes the top value on the Normal stack and prints it. The word LOOP increments the index value (on top of the Return stack) and then tests it with the limit value (second value on the Return stack). If the new index value is less than the limit, then control returns to the first word after the DO. Otherwise, the index and limit are popped from the Return stack and control passes out of the loop. The output produced by execution of the word EX2 is:

```
                      0 1 2 3
```

Note that the limit gives the number of times the loop is executed when the initial index is set to zero (0). The range of a DO loop is always executed at least once. Since DO loops use the Return stack, care must be taken if the words within the loop also use the Return stack.

Loops that increment the index by a value other than +1 are accomplished with the +LOOP word. +LOOP is like LOOP, except that the current Normal stack value is used to determine the new index. If the current Normal stack value is negative, then looping continues until the new index becomes less than the limit value.

Forward conditional branches may be made in 6800FORTH using the words IF, THEN, and ELSE. These words are more easily explained through the use of an example:

```
          : EX3 IF (true-words) ELSE (false words) THEN ;
```

When EX3 is executed, the word IF tests, and removes, the current stack value. If the value is non-zero, then the "true-words" are executed. If the value is zero (0) then the "false-words" are executed. The word THEN indicates the end of the IF...ELSE construct. It is always required. The word ELSE is optional. For example:

```
                    : EX4 IF (true-words) THEN ;
```

EX4 will cause the "true-words" to be executed only if the top stack value is non-zero.

10. ERROR CHECKING.

6800FORTH produces various error indications. The standard FORTH error is a ? (question mark). This is usually caused by a word in the input stream that cannot be found in the dictionary. Another error is "ERROR nn" where nn is an identification of the error. When either message is produced, 6800FORTH stops its current activity and returns to the input mode. If a word was in the process of being defined, the dictionary is returned to its state prior to the beginning of the definition. The stacks are re-initialized. The effect is vary similar to a 6800FORTH "Softstart".

MESSAGE		EXPLANATION
word ?	-	The word cannot be found in the dictionary, or it's use is illegal (e.g. ";" found while not in ":" mode).
nnnn ?	-	The number cannot be converted to a 16 bit binary value using the current base.
ERROR 00	-	Return stack overflow. Too many values pushed onto the Return stack or too many levels of DO nesting.
ERROR 01	-	Return stack underflow. A word was executed that pulls a value from the Return stack, but the value wasn't entered prior to the word's execution.
ERROR 02	-	Normal stack underflow. Same as for Return stack.
ERROR 03	-	Normal stack / dictionary cross. Too many values on the Normal stack causing it to overlay the dictionary. Since the dictionary and Normal stack grow toward each other, this error occurs more frequently as the dictionary becomes large. The dictionary may be damaged.
ERROR 04	-	Keyboard buffer (or Block buffer during EXEC) exceeded its upper boundry.
ERROR 05	-	Block buffer 0 exceeded its upper boundry.
ERROR 06	-	Block buffer 1 exceeded its upper boundry.
ERROR 09	-	A word containing an IF word was executed, but no THEN word was used. THEN is always required when IF is used.

Errors 00 through 06 will only be produced if the memory locations of the stacks, buffers, and dictionary are allowed to default. If the user initializes any of these locations (other than memory size) prior to a "Hardstart", these tests are not made.

11. STANDARD VOCABULARY.

Following each word difination is a graphic demonstration of the word's effect on the Normal stack. (1 2 3 4 word 1 2 3) shows that the values 1, 2, 3, and 4 were entered, with 4 on top of the stack. Then "word" was entered. In this case "word" caused the top stack value to be "popped" leaving the 3 as the top value. Examples using A000 indicate an address on the stack.

TYPING WORDS

.	Convert and type the signed value on top of the stack according to the current radix. (1 2 3 4 . 1 2 3)
#DIG	A variable whose value sets the tabulation stops for the word "." (period). The number of spaces and digits typed is equal to the value of #DIG. If #DIG is too small to allow complete typing of a number, it is ignored. (stack is un-affected)
.#	Types the unsigned value on top of the stack (Usually in HEX). For typing addresses, machine instructions, etc. (1 2 3 4 .# 1 2 3)
?	Uses the top of the stack as an address; and types the value at that address according to the current radix. (1 2 3 a000 ? 1 2 3)
ECHO	Types the LSB on top of the stack in ASCII. If the character is non-printing, a space will usually be printed. (1 2 3 4 ECHO 1 2 3)
SPACE	Types a single space. (stack is un-affected)
MSG	Types a string of characters until a $04 HEX value is found. The top stack value provides the address of the first byte of the string to be typed. Control characters may be imbedded in the string. (1 2 3 4 MSG 1 2 3)
"	Is used in a ":" (colon) definition of a word that will, when executed, place an address on the stack. The address points to a string of characters that is followed by a $04 HEX value. The string of characters is included in the definition and follows the word being defined. It is enclosed in " (quotes). The first blank following " (quote) is a 6800FORTH requirement and will not be included in the character string. EXAMPLE: : MSG1 " TEST MESSAGE" ;
TYPE	Types a string of characters whose address is found as the second value on the stack. The top stack value contains the number of characters to be typed. (1 2 A000 4 TYPE 1 2)

CR	Output a carriage return. (Stack is un-affected)
CLR	Clears a CRT display device to blanks. CLR should not be used for hard-copy devices. (stack is un-affected)
?BASE	Types the current radix using Decimal as a temporary radix for conversion of the typed number. (stack is un-affected)
DICT	Types every word in the 6800FORTH dictionary in the order it is searched. The address of each word's header is typed after each word. (stack is un-affected)

BLOCK I/O WORDS

BUFL	Places the low address of the screen buffer on the stack. (1 2 3 4 BUFL 1 2 3 4 E000)
BUFH	Places the high address of the screen buffer on the stack. (1 2 3 E000 BUFH 1 2 3 E000 E200)
LDBF	Initializes the cassette I/O routines with the beginning and ending values, of the block to be output, using the second and top values from the stack. (1 2 E000 E200 LDBF 1 2)
SAVE	Writes a buffer to cassette tape. The top value on the stack is the ID of the screen for subsequent retrieval. SAVE will replace itself in the buffer with the ";S" word; then clear everything after the ";S" word to blanks. (1 2 3 4 SAVE 1 2 3)
KEEP	Combines BUFL BUFH LDBF and SAVE to write the current screen to cassette tape. KEEP uses the top value on the stack for the screen ID for subsequent retrieval. (1 2 3 4 KEEP 1 2 3)
LOAD	Reads a block from the cassette tape into the screen buffer. The top value on the stack is the ID of the block to be read. (1 2 3 4 LOAD 1 2 3)
EXEC	Causes the contents of the current screen to be scanned and passed through the key-in routines just as if it was being received from the keyboard. The scan begins with the start of the screen and continues until the ";S" word is found. When EXEC is used, the screen should only contain word definitions followed by a single ";S" word. (stack is un-affected)
;S	Used in conjunction with EXEC to indicate the end of the scan. ;S returns 6800FORTH to the normal input mode. (stack is un-affected)

WORD-DEFINING WORDS

:	Begins a colon definition. The next word in the input is taken as the name of the new word. The interpreter is automatically set to the Compile mode. (stack is un-affected)
;	Terminates the colon definition, places a "Return to subroutine" (RTS) instruction in the dictionary, then switches the system back to Interpret mode. This word should only be used while in Compile mode. (stack is un-affected)
[	Used within ":" (colon) definitions to cause the words following to be executed rather then compiled. The effect is as if all words following the bracket were "Immediates". (stack is un-affected)
]	Used within ":" (colon) definitions after the "[" (left bracket). This word reverses the effect of the left bracket. Actually, both the left bracket and the right bracket words have the same effect on the system. Either will toggle the state of the system from Compile mode to Interpret mode, and vice versa. They are both included to allow the group of words, that are forced into the Immediate mode, to be set off in brackets. (stack is un-affected)
IMME	Used within ":" (colon) definitions just prior to the ";" word. It indicates, to the system, that the word which is being compiled should be identified as an "Immediate" word. An "Immediate" word is always executed when it is encountered in the input stream, regardless of the Compile/Interpret mode of the system. It will always be treated as if it were enclosed within brackets. (stack is un-affected)
CONS	Defines a word that will, when executed, push a constant value on the stack. The value of the constant is obtained from the top of the stack when CONS is executed. CONS is not normally used in a ":" (colon) definition. (1 2 3 4 CONS 1 2 3)
VARI	Defines a word that will, when executed, push an address on the stack. the address points to a value in the dictionary. The address can then be used to reference or change the value. When VARI is executed, the top stack value becomes the initial value of the valiable. VARI is not normally used in a ":" (colon) defination. (1 2 3 4 VARI 1 2 3)
ARRAY	Sets aside a region in the dictionary whose length (in 16 bit words) is the top stack value. The name of the array follows the word ARRAY in the input stream. When the new word is subsequently executed, the address of the first element of the array is pushed on the stack. None of the elements of the array are initialized. (1 2 3 4 ARRAY 1 2 3)

"	Is used in a ":" (colon) definition of a word that will, when executed, place an address on the stack. The address points to a string of characters that is followed by a $04 HEX value. The string of characters is included in the definition and follows the word being defined. It is enclosed in " (quotes). The first blank following " (quote) is a 6800FORTH requirement and will not be included in the character string. EXAMPLE: : MSG1 " TEST MESSAGE" ;

CONSTANTS.

BASE	Address of the current radix (or base) value. (1 2 3 4 BASE 1 2 3 4 62)
H	Address of the pointer to the next available dictionary location. (1 2 3 4 H 1 2 3 4 5A)
SP	Address of the pointer to the current (or top) Normal stack value. (1 2 3 4 SP 1 2 3 4 54)
RS	address of the pointer to the current (or top) Return stack value. (1 2 3 4 RS 1 2 3 4 84)

NORMAL STACK OPERATIONS.

DROP	Removes (or POPS) the current Normal stack value. The second stack value becomes the current value. (1 2 3 4 DROP 1 2 3)
DOWN	Pushes the Normal stack. The current stack value becomes the second stack value. The new current stack value is unpredictable. (1 2 3 4 DOWN 1 2 3 4 5)
DUP	Duplicates the current, Normal stack value. After DUP, the current and second stack values are the same. (1 2 3 4 DUP 1 2 3 4 4)
OVER	Duplicates the second stack value and pushes it on the stack. The current Normal stack value becomes the second stack value and the top and third values are the same. (1 2 3 4 OVER 1 2 3 4 3)
ROT	Moves the third stack value to the top of the stack. The top stack value becomes the second and the second stack value becomes the third. (1 2 3 4 ROT 1 3 4 2)
SWAP	Interchanges the top Normal stack value with the second value. (1 2 3 4 SWAP 1 2 4 3)

RETURN STACK OPERATIONS.

DRP. Removes the current Return stack value. The second Return stack value becomes the current value. (Normal stack is un-affected)

DWN. Pushes the Return stack. The current Return stack value becomes the second Return stack value. The new, current, Return stack value is unpredictable. (Normal stack is un-affected)

<R Removes the current Normal stack value and pushes it on the Return stack. (1 2 3 4 <R 1 2 3)

R> Removes the current Return stack value and pushes it on the Normal stack. (1 2 3 4 R> 1 2 3 4 5)

I Pushes the current Return stack value on the Normal stack without removing it from the Return stack. After "I" the current values on both stacks are the same. (1 2 3 4 I 1 2 3 4 5)

DICTIONARY OPERATIONS.

HERE Pushes the address of the next available dictionary location on the Normal stack. (1 2 3 4 HERE 1 2 3 4 A000)

TRUNC Truncates (or deletes) the dictionary beginning with the address that is found on the top of the Normal stack. The address must be six higher than the header of a word in the dictionary. (1 2 3 4 TRUNC 1 2 3)

FORGET Truncates (or deletes) the dictionary beginning with the word whose name follows the word FORGET. (stack is un-affected)

, Places the current Normal stack value (16 BITS) into the next two available dictionary locations. (1 2 3 4 , 1 2 3)

C, Places the LSB (8 BITS) of the current Normal stack value into the next available dictionarY location. (1 2 3 4 C, 1 2 3)

D= Places the machine's accumulator (Register A) into the next available dictionary location (8 BITS). (stack is un-affected)

CONDITIONAL BRANCHES.

None of the following group of words can be used outside of a ":" (colon) definition. The stack descriptions show the effect on the stack when the word, that is being defined, is subsequently executed. It is not the effect during compilation.

BEGIN Starts a loop which will be terminated by END. BEGIN can only be used within a ":" (colon) definition. (stack is un-affected)

END Terminates a BEGIN/END loop. END can only be used within a ":" (colon) definition. When the word that is being defined is subsequently executed, the code generated by END will pop the current, Normal stack value and test it for zero. If it is zero, a branch will be taken back to BEGIN.
(1 2 3 4 END 1 2 3)

DO Starts a loop which will be terminated by either LOOP or +LOOP. DO can only be used within a ":" (colon) defination. When the word that is being defined is subsequently executed, the DO uses the current stack value for the loop index; and and the second stack value as the final loop index. These values are pushed onto the Return stack so that the current, Return stack value is the loop index. (1 2 3 4 DO 1 2)

LOOP Terminates a DO/LOOP construct. Execution of the LOOP word will add one to the loop index. If the new index value is less than the final index value (second value on the Return stack), then control is transfered to the word following the DO, and the loop is repeated. Otherwise, the two index values are popped from the Return stack and the looping sequence ends. The DO/LOOP construct is only used within ":" (colon) definitions. (Normal stack is un-affected)

+LOOP Terminates a DO/+LOOP construct, and is exactly like the DO/LOOP previously defined. The exception is that the current, Normal stack value is added to the loop index (on the Return stack) to form the new loop index. If the current, Normal stack value is negative, looping will continue while the new index value is greater than the final value (second value on the Return stack). The DO/+LOOP construct is only used within ":" (colon) definitions. (1 2 3 4 +LOOP 1 2 3)

IF Starts an IF/THEN conditional branch. The IF word, when executed, tests (and removes) the current, Normal stack value. If the value is not equal to zero, then the clause immediately following the IF is executed. If the value is zero, then the clause following the ELSE is executed. If the ELSE is omitted, then control is given to the clause following the THEN. THEN is always required with the IF word. IF can only be used within ":" (colon) definitions. (1 2 3 4 IF 1 2 3)

ELSE Indicates the beginning of a false (equal to zero) clause in an IF/ELSE/THEN construct. ELSE is only used within ":" (colon) definitions, and only with IF. Its use with IF is optional. (stack is un-affected)

THEN Terminates the IF/THEN construct. Words following THEN are executed without respect to the preceeding IF conditional test. THEN is only used within ":" (colon) definitions, and only with IF. IF cannot be used without a terminating THEN. If a word is defined using IF without THEN, ERROR 09 will be printed when that word is subsequently executed. (stack is un-affected)

ADDRESS OPERATORS.

@ Uses the current stack value as an address, and places the 16 BIT value at that address on the stack. The address that was on the stack is replaced by its value. (1 2 3 A000 @ 1 2 3 4)

C@ Uses the current stack value as an address and places the 8 BIT LSB of the 16 BIT value at that address on the stack. The address that was on the stack is replaced by the 8 BIT value. (1 2 3 A000 C@ 1 2 3 4)

! Uses the current stack value as an address and stores the second (16 BIT) stack value at that address.(1 2 3 A000 ! 1 2)

C! Uses the current stack value as an address and stores the (8 BIT) LSB of the second stack value at that address. (1 2 3 A000 C! 1 2)

+! Uses the current stack value as an address. The value at that address is replaced by the sum of itself and the second stack value. Both the address and the addend are removed from the stack. (1 2 3 A000 +! 1 2)

LOGICAL OPERATORS.

AND Performs a logical "AND" with the top two stack values. The two values are replaced, on the stack, by the result. (1 2 3 4 AND 1 2 0)

OR Performs a logical "OR" with the top two stack values. The two values are replaced, on the stack, by the result. (1 2 3 4 OR 1 2 7)

XOR Performs a logical "EXCLUSIVE OR" with the top two stack values. The two values are replaced, on the stack, by the result. (1 2 3 4 XOR 1 2 7)

SWAB Exchanges the LSB with the MSB of the current, stack value. (1 2 3 4 SWAB 1 2 3 400)

TEST OPERATORS.

0=	Tests the current stack value for zero. If the value is zero, it is replaced on the stack with the value one (1). A non-zero value is replaced with the value zero. (1 2 3 4 0= 1 2 3 0)
NOT	Identical to 0= (above). (1 2 3 4 NOT 1 2 3 0)
0<	Tests the current stack value for negative. If the value is negative, it is replaced on the stack with the value one (1). A positive (or zero) value is replaced by the value zero. (1 2 3 4 0< 1 2 3 0)
=	Tests the top two stack values for equality. If they are equal, both values are replaced by the value one (1). Otherwise, both values are replaced by the value zero. (1 2 3 4 = 1 2 0)
>	Tests the second stack value for greater than the top stack value. A signed comparison is used. If the second stack value is greater, both values are replaced by the value one (1). Otherwise, both values are replaced by the value zero. (1 2 3 4 > 1 2 0)
<	Tests the second stack value for less than the top stack value. A signed comparison is used. If the second stack value is less, both values are replaced by the value one (1). Otherwise, both values are replaced by the value zero. (1 2 3 4 < 1 2 1)
MIN	Compares the top two stack values and keeps only the smaller value. The greater value (or top value if equal) is removed from the stack. A signed comparison is used. (1 2 3 4 MIN 1 2 3)
MAX	Compares the top two stack values and keeps only the larger value. The smaller value (or top value if equal) is removed from the stack. A signed comparison is used. (1 2 3 4 MAX 1 2 4)

ARITHMETIC OPERATORS.

+	Adds the top two stack values and replaces them, on the stack, with their sum. (1 2 3 4 + 1 2 7)
1+	Increments the top stack value by 1. (1 2 3 4 1+ 1 2 3 5)
-	Subrtacts the top stack value from the second stack value and replaces them, on the stack, with their difference. (1 2 3 4 - 1 2 -1)

*	Multiplies the top two stack values and replaces them, on the stack, with their 16 BIT product. (1 2 3 4 * 1 2 12)
2*	Doubles the top stack value. (1 2 3 4 2* 1 2 3 8)
/MOD	Divides the second (16 BIT) stack value by the top (16 BIT) stack value. The second stack value (dividend) is replaced by the (16 BIT) quotient. The top stack value (divisor) is replaced by the (16 BIT) remainder. (1 2 5 3 /MOD 1 2 1 2)
MINUS	Changes the sign of the top stack value. (1 2 3 4 MINUS 1 2 3 -4)
ABS	Replaces the top stack value with its absolute value. (Negative values are made positive; positive values are left alone). (1 2 3 -4 ABS 1 2 3 4)
DECI	Changes the radix (or base) to decimal. Arithmetic operations and ASCII conversions are affected by the value of the radix. (stack is un-affected)
HEX	Changes the radix (or base) to hexadecimal. Arithmetic operations and ASCII conversions are affected by the value of the radix. (stack is un-affected)
OCTA	Changes the radix (or base) to octal. Arithmetic operations and ASCII conversions are affected by the value of the radix. (stack is un-affected)

MISCELLANEOUS WORDS.

'	The dictionary is searched for the word that immediately follows the ' (quote). When that word is found, the address of the executable machine code (following its header) is placed on the stack. (1 2 3 4 ' word 1 2 3 4 A000)
(	Begins a string of text that will be skipped by the 6800FORTH interpreter. The string being ignored must be terminated by a ")". The first blank following the "(" is a 6800FORTH requirement and cannot be omitted. (stack is un-affected)
BYE	Returns control to the machine's monitor.
FSE	Places the 6800FORTH system in "FORTH SCREEN EDIT" mode. The effect is that all keyboard input is ignored. The keyboard buffer remains empty. This allows the cursor controls to be used to edit a screen without filling the keyboard buffer with partial word definitions. FSE mode remains in effect until the first carriage return. (stack is un-affected)

RNDM	Generates a random number. The top value on the stack is used to determine the upper limit for the number. The number generated will be positive and less than the top stack value. The top stack value will be replaced by the random number. (1 2 3 4 RNDM 1 2 3 1)
ABIT	Holds the 6800FORTH system in a tight loop for about a quarter of a second. It is used in conjunction with a DO/LOOP or BEGIN/END loop to allow a pause during word execution. (stack is un-affected)
SECO	Uses the top value on the stack as the number of iterations for a loop containing four ABITs. The effect of "5 SECONDS" would be a delay of approximately 5 seconds. (1 2 3 4 SECO 1 2 3)
$NEW	Prepares a 6800FORTH program, including a newly expanded dictionary, for output by the machine monitor's output facility. $NEW updates the dictionary initialization words to include any newly added words. It also clears all of the boundry initialization words. After $NEW, the 6800FORTH system must be "Hardstarted". For this reason, $NEW will give control to the machine's monitor.
ZERO	Clears an area to Hex zeros. The second stack value is used as the beginning address of the area to be cleared. The top stack value is the number of 8-BIT locations to be cleared. (1 2 A000 4 ZERO 1 2)

12. 6800FORTH ASSEMBLER

A large number of the words in the 6800FORTH dictionary closely resemble the 6800 mnemonic instruction set. They are used to generate 6800 machine code and add it to the dictionary. For this reason, they need to be executed immediately (i.e. during the compilation of some other word). To cause them to be executed immediately, they usually follow the "[" (left bracket).

Although the mnemonics are almost the same, the syntax of the instructions is quite different. The traditional "OPERATOR followed by OPERAND" format has been reversed to accomodate the 6800FORTH syntax. This allows the operands to be pushed onto the stack and then operated upon by the mnomonic OP codes. The use of the stack also requires that the 6800FORTH assembler words be informed of the desired addressing mode of those instructions that contain addresses. The 6800FORTH assembler format is as follows:

OPERAND   MNEMONIC   ADDRESSING MODE

All of the 6800 mnemonic instruction set has been included in the 6800FORTH dictionary. Those mnemonics that could be confused with valid HEX numbers or with other valid 6800FORTH words (e.g. ADDA ADCB BCC CLR) have been altered slightly to avoid the confusion. (e.g. ADA, ACB, BCC, CLR,) The alteration, in all cases involves adding a comma. The following page contains a list of all of the 6800FORTH assembler mnemonic OP codes. The * (asterisk) following some words is not part of the word. It indicates which of the words requires an addressing mode identifier.

The list is divided into three sections. The words in the first section require an operand on the stack and an addressing mode identifier following the word. This group of words will add either 2 or 3 bytes of machine code (depending on the addressing mode and instruction) to the dictionary.

The second group of words all imply the relative addressing mode, so no mode identifier is used. This group of words also finds its operands on the stack. Two (2) bytes of machine code will be added to the dictionary by any of these words.

The third group of words are implicit, and require neither an operand nor an addressing mode identifier. each of these words will add 1 byte of machine code to the dictionary when executed.

GROUP 1 (Format: OPERAND MNEMONIC MODE)

ACA, *	ACB, *	ADA, *	ADB, *
ANDA *	ANDB *	ASL *	ASR *
BITA *	BITB *	CLR, *	CMPA *
CMPB *	COM *	CPX *	DEC, *
EORA *	EORB *	INC *	JMP *
JSR *	LDAA *	LDAB *	LDS *
LDX *	LSR *	NEG *	ORAA *
ORAB *	ROL *	ROR *	SBCA *
SBCB *	STAA *	STAB *	STS *
STX *	SUBA *	SUBB *	TST *

GROUP 2 (Format: OPERAND MNEMONIC)

BCC,	BCS	BEQ	BGE
BGT	BHI	BLE	BLS
BLT	BMI	BNE	BPL
BRA	BSR	BVC	BVS

GROUP 3 (Format: MNEMONIC)

ABA,	ASLA	ASLB	ASRA
ASRB	CBA,	CLC	CLI
CLRA	CLRB	CLV	COMA
COMB	DAA,	DCA,	DCB,
DES	DEX	INCA	INCB
INS	INX	LSRA	LSRB
NEGA	NEGB	NOP	PSHA
PSHB	PULA	PULB	ROLA
ROLB	RORA	RORB	RTI
RTS	SBA	SEC	SEI
SEV	SWI	TAB	TAP
TBA	TPA	TSTA	TSTB
TSX	TXS	WAI	

* The three addressing mode identifier words are:

#	Immediate
,X	Indexed
@#	Direct/Extended (Direct is used if the address value, on the stack, is less than 256. The system will make this determination.)

It is the responsibility of the user to insure that only valid addressing modes are used.

The 6800FORTH assembler words are executed by the system just like any other word. Unlike most other words, the assembler words will add data to the dictionary when they are executed. The words:

```
5 LDAA
```

will cause two bytes (86 05) to be added to the dictionary. This is undesirable unless a word is being defined. For this reason, the assembler words should only be used within ":" (conon) definitions, and then, only within the "[" and "]" (bracket) words. For example:

```
KILL [ 0 LDX # 0 CLR, ,x INX FB BRA ;
```

defines a word (KILL) in the dictionary. The ":" and "KILL" words are the standard beginning of a colon definition. They cause a header to be added to the dictionary. (the description of the dictionary on pages 5 and 6 will explain the header in detail). Following the header is always some machine executable code. Usually that code consists of one or more JSR (jump-to subroutine) instructions. In this example the "[" (bracket) word causes the words that follow to be executed immediately. Since the words that follow the "[" add data to the dictionary when they are executed, they will cause seven bytes (CE00 7e00 08 20fb) to be placed in the dictionary after the header for the word KILL. It is important to understand the effect of the "[" (bracket). Had it not been used, the machine executable code following the header for "KILL" would have been a list of nine JSR instructions. The JSR instructions, when executed, would add the same seven bytes (CE00 7E00 08 20FB) to the dictionary. The important difference is that the seven bytes would not be added unitl the word KILL was executed, and they would not be a part of the KILL word's definition.

This is an important distinction, and if used properly it can add a powerful MACRO extension to the 6800FORTH assembler. Words can be defined which will add machine executable code to the dictionary each time they are executed. Some of these MACRO words have been used in the development of 6800FORTH. They remain in the dictionary and can be of use to the user. they are as follows:

MACRO	CODE GENERATED	COMMENTS
X=SP	54 LDX @#	Loads the Index register with the address of the current stack value
SP=X	54 STX @#	Stores the address of the current stack value from the Index register
X=RS	84 LDX @#	Loads the Index register with the address of the current Return stack value
RS=X	84 STX @#	Stores the address of the current Return stack value from the Index Register

SP@	54 LDX @# 1 LDAA ,X 0 LDAB ,X	Loads the Index register with the address of the current stack value and the BA registers with that 16-BIT value
SP=	54 LDX @# 1 STAA ,X 0 STAB ,X	Loads the Index register with the address of the current stack value and stores the 16-BIT value from the BA registers at that address
RS@	84 ldx @# 1 LDAA ,X 0 LDAB ,x	Same as SP@ except for the Return stack
RS=	84 LDX @#	Same as SP= except for the Return stack